Rによる英文テキスト解析

著者	小林 雄一郎
著者別名	KOBAYASHI Yuichiro
雑誌名	東洋大学社会学部紀要
巻	53
号	1
ページ	51-64
発行年	2015-11
URL	http://id.nii.ac.jp/1060/00008219/

Rによる英文テキスト解析

Analyzing English Texts with R

小林雄一郎
Yuichiro　KOBAYASHI

1．はじめに

情報化時代と言われる現代において、大量のデータから情報や知識を効率よく取り出すためのデータマイニング技術が大きな関心を集めている。それにともない、言語データの分析にとっても、コンピュータが不可欠なものとなっている。かつての言語研究者は、紙に書かれた用例を1つずつ丹念に読んでいた。しかし、現代の大規模データを分析する場合、このような手作業は、多くの困難をともなう。例えば、1億語の言語データは、1日8時間声に出して読み続けたとしても、全てを読むのに4年間はかかると推定されている（Aston & Burnard, 1998, p. 28）。従って、手作業による分析は、単に退屈であるだけでなく、重要な用例を見落とす危険性すら孕んでいる（Hammond, 2002, p. 2）。

勿論、英文テキストを解析するためのGUIツールは、すでに数多く公開されている。しかし、既存のツールには多くのユーザーが利用する「最大公約数的な機能」しか搭載されておらず、自分に必要な機能が用意されているとは限らない（淺尾・李、2013, p. iii）。また、ユーザーフレンドリーなツールでは、データ処理の過程がブラックボックスとなっているために、出力結果の正しさを検証することが難しく、検証の必要性自体も意識されにくくなる（大名、2012, p. iii）。例えば、ツールごとに「単語」の定義が異なり（Anthony, 2013, p. 150）、異なるツールによって計算された「語数」が最大で10パーセントも異なること（Meunier, 1998, p. 27）は、一般のユーザーにはそれほど強く意識されていない。そして、既存のツールに依存することで、ツールの限界がそのまま研究の限界となり、ツールなしでは全く研究ができない研究者も存在し得る（Gries, 2010, p. 124）。

このような状況において、研究者の目的に合わせて、独自の解析プログラムを作成することが非常に有益であることは言うまでもない。自らプログラムを作ることで、（1）既存のツールではできない分析が可能になり、（2）検索の速度と精度が向上し、（3）自分の研究に使いやすいような出力を得られ、（4）データサイズの制約を受けずに分析できるようになる（Biber, Conrad, & Reppen, 1998, pp. 255–256）。

本稿の目的は、Rというプログラミング言語を用いて、英文テキストの用例検索や頻度集計を効率

的に行うための方法を提示することである。テキスト解析には、Python（淺尾・李、2013；Johnson, 2013）、 Perl（赤瀬川・中尾、2004；Hammond, 2003）、 Ruby（荻野・田野村、2012）、 JAVA（Hammond, 2002；Mason, 2001）， AWK（Schmitt, Christianson, & Gupta, 2010）， PHP（斉 藤・ 高 橋、2014）など、R 以外の言語が用いられることもある。それらの言語と比べて、R は統計処理と視覚化に優れている。今日の言語研究において、統計処理の重要性は言うまでもない。（1）前処理や頻度集計といったテキスト処理、（2）仮説検定や多変量解析のような統計処理、（3）様々なグラフ形式による解析結果の視覚化、を 1 つの環境で連続的に行えるメリットは大きい。

2．R とパッケージのインストール

R は、 Windows、 Mac、 Linux などの複数の OS で動作するマルチプラットフォームのプログラムであり、公式ウェブサイト（http://www.r-project.org/）から無償でダウンロードすることができる。R 本体のインストール方法や基本操作に関しては、Lander（2013）などに詳しい。

また、様々な解析に特化した追加パッケージも同じウェブサイトから入手可能である。そして、本稿で用いる追加パッケージは、 languageR、 tm、 wordcloud の 3 つである。これらをインストールするには、 R を起動し、インターネットに接続可能な環境で、以下のスクリプトを入力する。行頭の > は 1 つのコマンドの開始位置を示すものであり、自分で入力する必要はない。因みに、# で始まる部分はコメントであり、省略可能である。つまり、実際の処理では、ボールド体の部分のみを入力すればよい。

```
> # 追加パッケージのインストール
> install.packages(c("languageR", "tm", "wordcloud"), dependencies
= TRUE)
```

3．データの読み込み

本稿では、簡便のために、 languageR パッケージに収められている alice データセットを分析対象とする。これは、 Lewis Carroll の *Alice's adventure in wonderland*（1865）のテキストから句読点を除いたものである。このデータを読み込んで、その冒頭部分を確認するには、以下のスクリプトを入力する。

```
> # データの読み込み
> library(languageR)
> corpus <- alice
> # データの冒頭20語のみを表示
> head(corpus, 20)
```

上記のスクリプトを実行すると、以下のような結果が得られる。

```
 [1] "ALICE"       "S"           "ADVENTURES" "IN"
 [5] "WONDERLAND" "Lewis"       "Carroll"    "THE"
 [9] "MILLENNIUM" "FULCRUM"     "EDITION"    "3"
[13] "0"           "CHAPTER"     "I"          "Down"
[17] "the"         "Rabbit-Hole" "Alice"      "was"
```

因みに、Rのワーキングディレクトリに保存されているファイル（例えば、hoge.txt）を読み込むには、以下のようにする。

```
> # ローカルファイルの読み込み
> corpus2 <- scan("hoge.txt", what = "char", sep = "¥n", quiet = TRUE)
```

また、Project Gutenberg（https://www.gutenberg.org/）などのウェブサイトで公開されているテキストファイルを直接読み込む場合は、以下のようにURL（例えば、http://www.xxx/yyy.txt）を直接指定する。

```
> # インターネット上のデータの読み込み
> corpus3 <- scan("http://www.xxx/yyy.txt", what = "char", sep = "¥n",
quiet = TRUE)
```

そして、XML形式のファイルを読み込む方法についてはJockers（2014）、SGML形式のファイルを読み込む方法についてはGries（2009）をそれぞれ参照されたい。

次節以降、上段で読み込んだaliceデータを用いて、（1）KWICコンコーダンス、（2）ワードリスト、（3）コロケーションテーブル、（4）n-gramリスト、の作成を行う。これらの処理のうち、（1）から（3）に関しては、Gries（2009）のスクリプトに基づく。但し、本稿のスクリプトでは、数字、句読点、空白の削除などの前処理が追加されている。また、コンコーダンスプロットやワードクラウドなどの視覚化の方法を提示する。

4．KWIC コンコーダンス

テキストにおける特定の単語や表現に関して、その用例を網羅的に抽出する場合は、 KWIC（Key Word In Context）コンコーダンスという表示形式（Baker, Hardie, & McEnery, 2006, pp. 42–44）が一般的である。この形式を用いることで、検索語がどのような文脈で使われているかを確認することができる。以下の例では、 *rabbit* を検索語として、その前後 5 語ずつを表示している。また、 *rabbit* と *Rabbit* を一度に検索できるように、テキスト中の全ての文字を小文字に変換し、余分なスペースを削除している。

```
> # 大文字を小文字に変換
> corpus.lower <- tolower(corpus)
> # テキストを単語のベクトルに変換
> word.vector <- unlist(strsplit(corpus.lower, "¥¥W"))
> # スペースを削除
> not.blank <- which(word.vector != "")
> word.vector2 <- word.vector[not.blank]
> # 検索語の生起位置を取得
> word.positions <- which(word.vector2[] == "rabbit")
> # 検索語の前後何語まで表示するかを指定（以下の例では，5 語）
> context <- 5
> # KWIC コンコーダンスの作成
> for(i in 1:length(word.positions)){
>   start <- word.positions[i] - context
>   end <- word.positions[i] + context
>   before <- word.vector2[start : (start + context - 1)]
>   after <- word.vector2[(start + context + 1) : end]
>   keyword <- word.vector2[start + context]
>   cat("--------------------", i, "--------------------", "¥n")
>   cat(before, "[", keyword, "]", after, "¥n")
> }
```

上記のスクリプトを実行すると、以下のような結果が得られる。R の alice データセットの中で *rabbit* は51回使われているが、以下では、紙面の都合で最初の10例のみを示す。

```
-------------------- 1 --------------------
0 chapter i down the [ rabbit ] hole alice was beginning to
-------------------- 2 --------------------
daisies when suddenly a white [ rabbit ] with pink eyes ran close
-------------------- 3 --------------------
the way to hear the [ rabbit ] say to itself oh dear
-------------------- 4 --------------------
quite natural but when the [ rabbit ] actually took a watch out
-------------------- 5 --------------------
had never before seen a [ rabbit ] with either a waistcoat pocket
-------------------- 6 --------------------
it pop down a large [ rabbit ] hole under the hedge in
-------------------- 7 --------------------
to get out again the [ rabbit ] hole went straight on like
-------------------- 8 --------------------
long passage and the white [ rabbit ] was still in sight hurrying
-------------------- 9 --------------------
turned the corner but the [ rabbit ] was no longer to be
-------------------- 10 --------------------
coming it was the white [ rabbit ] returning splendidly dressed with
a
```

そして、テキスト中で *rabbit* が使われている位置を確認するには、コンコーダンスプロット（Anthony, 2005, p. 9）による視覚化が有効である。コンコーダンスプロットでは、テキスト中で検索語が表れている位置をバーコードのような形式で表現される。

```
> # 検索語の生起位置を視覚化
> plot(corpus.lower == "rabbit", type = "h", yaxt = "n", main =
"rabbit")
```

上記のスクリプトを実行すると、図1のような結果が得られる。それを見ると、*rabbit* が物語の前半と後半で多く使われていることが分かる。

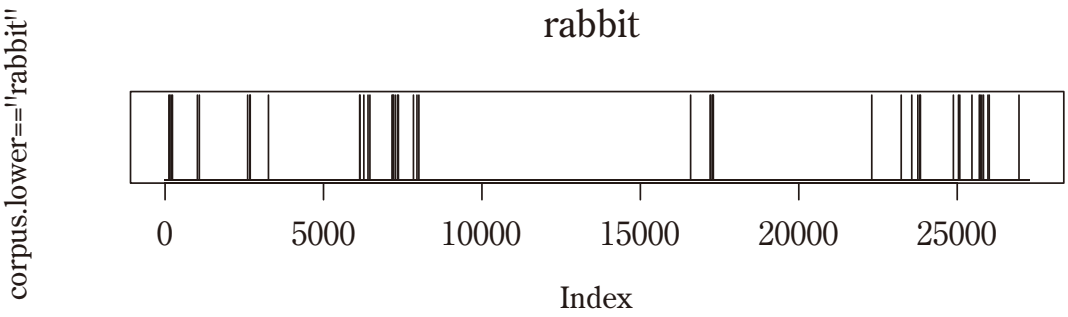

図1：コンコーダンスプロット

5．コロケーションテーブル

また、分析対象とする単語がどのような単語と一緒に使われているのかを知りたい場合には、コロケーション（共起語）（Baker, Hardie, & McEnery, 2006, pp. 36–38）の頻度を集計する。以下の例では、*rabbit* の前後3語の位置で使われている単語の頻度を集計し、その結果をコロケーションテーブルという形式で、出力ファイル（output.txt）に保存している。

```
> # ノードワードの指定（以下の例では，"rabbit"）
> search.word <- "¥¥brabbit¥¥b"
> # スパンの指定（以下の例では，前後3語まで）
> span <- 3
> span <- (-span : span)
> # 出力ファイルの指定
> output.file <- "output.txt"
> # ノードワードの出現する位置を特定
> positions.of.matches <- grep(search.word, word.vector2, perl =
TRUE)
> # コロケーションの集計
> results <- list()
> for(i in 1 : length(span)) {
>      collocate.positions <- positions.of.matches + span[i]
>      collocates <- word.vector2[collocate.positions]
>      sorted.collocates <- sort(table(collocates), decreasing =
TRUE)
>      results[[i]] <- sorted.collocates
> }
```

```
> # 集計表のヘッダーを出力
> cat(paste(rep(c("W_", "F_"), length(span)), rep(span, each = 2),
sep = ""), "¥n", sep = "¥t", file = output.file)
> # 集計データを出力
> lengths <- sapply(results, length)
> for(k in 1 : max(lengths)) {
>      output.string <- paste(names(sapply(results, "[", k]),
sapply(results, "[", k], sep = "¥t")
>      output.string.2 <- gsub("NA¥tNA", "¥t", output.string, perl
= TRUE)
>      cat(output.string.2, "¥n", sep = "¥t", file = output.file,
append = TRUE)
> }
```

上記のスクリプトを実行すると、表1のようなコロケーションテーブルが得られる。表中のW_0は検索語を表し、W_−3、W_−2、W_−1、W_1、W_2、W_3は、検索語から見て左3語の位置、左2語の位置、左1語の位置、右1語の位置、右2語の位置、右3語の位置、をそれぞれ表している。また、F_−3～F_3は、左3語～右3語の位置に生起する語の頻度を表している。表1を見ると、*rabbit* の直前（W_−1）では *the* や *white*、直後（W_1）では *hole* などが多く使われていることが分かる。

6．ワードリスト

次に、テキスト中に現れている全ての単語の頻度を集計し、ワードリスト（語彙頻度表）（Baker, Hardie, & McEnery, 2006, p. 169）を作成する。以下の例では、 tmパッケージのremoveNumbers関数とremovePunctuation関数を用いて、テキスト中の数字と句読点を削除している（aliceデータに句読点は含まれていないが、一般的なテキストには句読点が多く含まれている）。

```
> # テキストから数字と句読点を削除
> library(tm)
> corpus.cleaned <- removeNumbers(corpus.lower)
> corpus.cleaned <- removePunctuation(corpus.cleaned)
> # ワードリストの作成
> freq.list <- table(corpus.cleaned)
> sorted.freq.list <- sort(freq.list, decreasing = TRUE)
```

表１：コロケーションの集計結果（一部）

W_−３	F_−３	W_−２	F_−２	W_-１	F_-１	W_０	F_０	W_１	F_１	W_２	F_２	W_３	F_３
said	4	the	21	the	24	rabbit	51	hole	4	a	3	a	5
was	3	said	3	white	22			s	4	alice	3	in	3
and	2	a	2	a	2			with	3	was	3	the	3
down	2	down	2	large	1			and	2	in	2	blasts	2
it	2	heard	2	that	1			blew	2	no	2	there	1
when	2	when	2	w	1			came	2	out	2	along	1
a	1	again	1					in	2	three	2	and	1
alice	1	as	1					read	2	to	2	as	1
angry	1	but	1					say	2	under	2	at	1
as	1	ears	1					was	2	up	2	back	1
at	1	for	1					who	2	voice	2	barrowful	1
before	1	hear	1					actually	1	and	1	come	1
bill	1	hush	1					angrily	1	by	1	dinah	1
but	1	iv	1					as	1	either	1	dressed	1
buy	1	name	1					asked	1	fact	1	dropped	1
chapter	1	of	1					began	1	had	1	eyes	1
corner	1	pity	1					but	1	he	1	fumbled	1
for	1	presently	1					coming	1	here	1	gave	1
gone	1	seen	1					cried	1	interrupted	1	her	1
hush	1	sir	1					engraved	1	it	1	his	1

```
> sorted.table <- paste(names(sorted.freq.list), sorted.freq.list,
sep = ": ")
> # ワードリスト（頻度上位20位まで）の確認
> head(sorted.table, 20)
```

alice データセットにおける頻度上位20語は、以下の通りである。一般的に、頻度上位語の多くは、*the* や *and* などの機能語である。どのようなテキストにも高頻度で現れる語を集計対象から除外したい場合は、tm パッケージの removeWords 関数で細かく指定することができる。

```
 [1] "the: 1639" "and: 866"   "to: 725"    "a: 631"
 [5] "it: 595"   "she: 553"   "i: 545"     "of: 511"
 [9] "said: 462" "you: 411"   "alice: 398" "in: 367"
[13] "was: 357"  "that: 315"  "as: 263"    "her: 248"
[17] "t: 218"    "at: 212"    "s: 201"     "on: 193"
```

そして、テキスト中に現れている全ての単語をワードクラウド（Abedin & Das, 2015, pp. 101–103）の形式で視覚化するには、wordcloud パッケージの wordcloud 関数を用いる。以下の例では、使用頻度が5回以上の単語のみを表示している（図2）。ワードクラウドでは、高頻度語が大きなフォントで表示され、低頻度語は小さなフォントで表示されている。

```
> # ワードクラウドの作成
> library(wordcloud)
> wordcloud(word.vector2, min.freq = 5, random.order = FALSE)
```

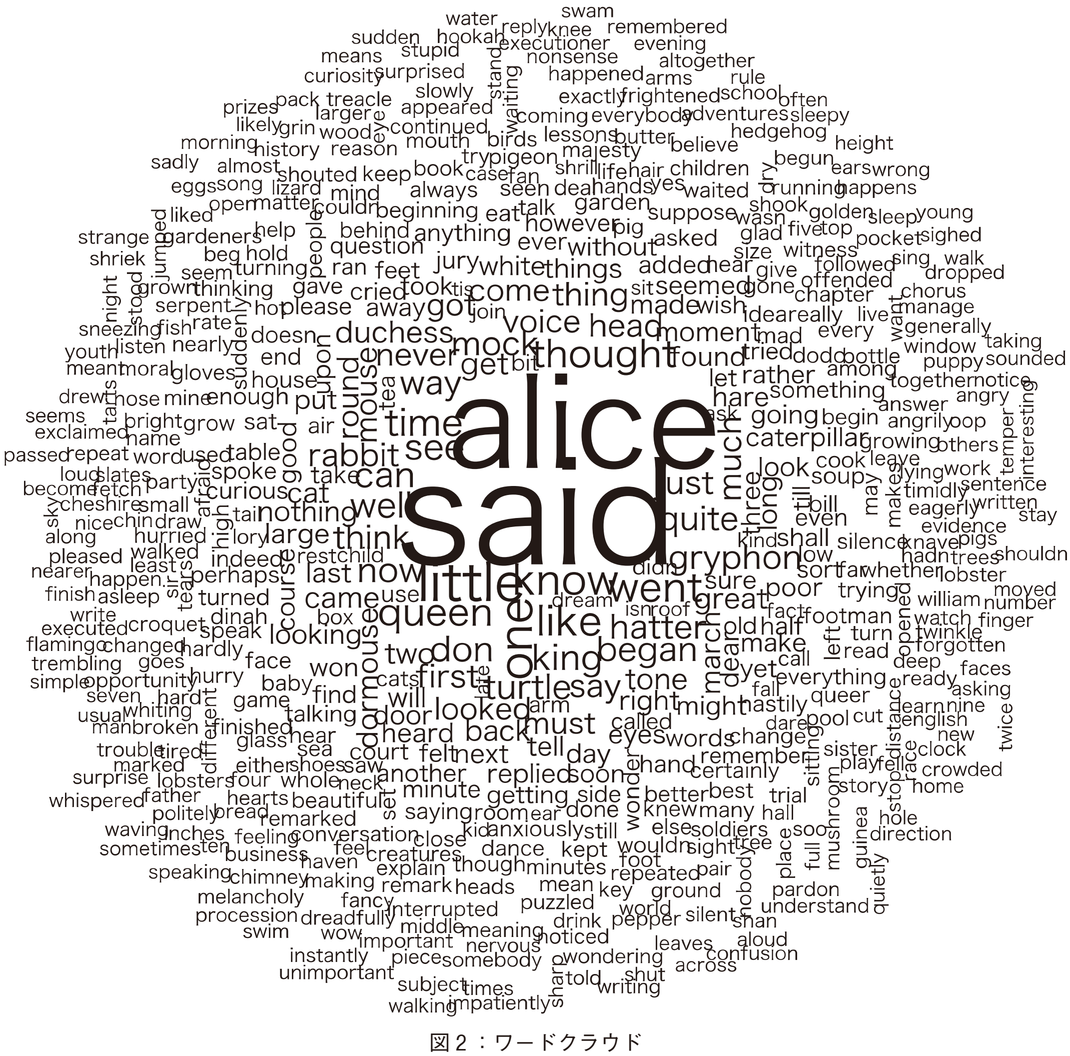

図２：ワードクラウド

また、テキストにおける語彙の豊富さを測る指標としては、 type/token ratio（TTR）（Baker, Hardie, & McEnery, 2006, p. 162）が最も一般的である。

```
> # 総語数（tokens）の計算
> length(corpus.cleaned)
> # 異語数（types）の計算
> length(unique(corpus.cleaned))
> # TTR の計算
> length(unique(corpus.cleaned)) / length(corpus.cleaned)
```

上記のスクリプトによって計算された総語数、異語数、 TTR は、以下の通りである。TTR の値が

1に近いほど、テキスト中の語彙が豊富であることを示している。

```
[1] 27269  #総語数
[1] 2604  #異語数
[1] 0.09549305  # TTR
```

なお、 koRpusパッケージを用いることで、語彙の豊富さに関する様々な指標を計算することが可能である（小林、2015）。

さらに、ステミング（語幹処理）（Bird, Klein, & Loper, 2009, p. 107）を行う場合は、 tmパッケージのstemDocument関数が便利である。しかし、レマタイズ（見出し語化）（Bird, Klein, & Loper, 2009, p. 108）を行うには、 TreeTagger（http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/）のような外部のプログラムを使う必要がある。

7．n-gramリスト

最後に、テキスト中における単語連鎖（n-grams）（Baker, Hardie, & McEnery, 2006, p. 122）を集計する。以下は、 2語の連鎖（2-grams）の頻度を集計し、頻度上位20位までを表示している。

```
> # 2-gramsの抽出
> x <- length(word.vector2) - 1
> results <- vector()
> for (i in 1 : x) {
>      ngram <- paste(word.vector2[i], word.vector2[i + 1])
>      results <- append(results, ngram)
> }
> # 頻度集計
> ngram.freq <- table(results)
> sorted.ngram.freq <- sort(ngram.freq, decreasing = TRUE)
> sorted.table <- paste(names(sorted.ngram.freq), sorted.ngram.freq,
sep = ": ")
> # 頻度上位20位までを表示
> head(sorted.table, 20)
```

上記のスクリプトを実行すると、以下のような集計結果が得られる。

```
 [1] "said the: 210"  "of the: 133"    "said alice: 116"
 [4] "in a: 97"       "and the: 82"    "in the: 80"
 [7] "it was: 76"     "the queen: 72"  "to the: 69"
[10] "the king: 62"   "as she: 61"     "don t: 61"
[13] "at the: 60"     "she had: 60"    "a little: 59"
[16] "i m: 59"        "it s: 57"       "mock turtle: 56"
[19] "and she: 55"    "she was: 55"
```

また、スクリプトを若干修正することで、3語の連鎖（3-grams）や4語の連鎖（4-grams）の頻度を集計することも可能である。なお、単純な単語連鎖であれば、 ngram パッケージを用いて集計することもできる。

8．おわりに

データに基づく実証的研究には、再現性が求められる。しかしながら、前述のように、既存のツールを複数使用すると、同じ入力データから異なる出力結果が得られることも多い。従って、研究者自身がデータ処理の過程を理解し、処理方法の違いが解析結果に及ぼす影響について自覚的になる必要がある。そして、データ処理に対する知識を深めることで、既存のツールを使う際にも、ブラックボックス化された処理の内容をある程度は推測できるようになり、誤った使い方を避けることがきできる（大名、2012, p. iv）。

本稿では、Rによる（1）KWICコンコーダンス、（2）ワードリスト、（3）コロケーションテーブル、（4）n-gramリスト、の作成方法を提示した。無論、プログラミングにおいては、同じ目的を達成するための複数の方法が存在する。（自分にとって）より良いRのプログラムを書くには、 Matloff（2011）や間瀬（2014）などが参考になる。また、集計した単語やn-gramの頻度を統計的に処理する方法については、 Baayen（2008）、 Gries（2013）、石田・小林（2013）などを参照されたい。そして、Rの視覚化機能については、Sarkar（2008）やWickham（2009）などに詳しい。

参考文献

Abedin, J., & Das, K. K.（2015）. *Data manipulation with R.* Second edition. Birmingham: PACKT Publishing.
赤瀬川史郎・中尾浩（2004）.『コーパス言語学の技法―言語データの収集とコーパスの構築』東京：夏目書房.
Anthony, L.（2005）. AntConc: A learner and classroom friendly, multi-platform corpus analysis toolkit. In Anthony, L., Fujita, S., & Harada, Y.（Eds.）, *Proceedings of IWLeL 2004: An Interactive Workshop on Language e-Learning*（pp. 7−13）.
Anthony, L.（2013）. A critical look at software tools in corpus linguistics. *Linguistic Research,* 30（2）, 141−161.
淺尾仁彦・李在鎬（2013）.『言語研究のためのプログラミング入門―Pythonを活用したテキスト処理入門』東

京：開拓社.
Aston, G., & Burnard, L.（1998）. *The BNC handbook: Exploring the British National Corpus with SARA.* Edinburgh: Edinburgh University Press.
Baayen, R. H.（2008）. *Analyzing linguistic data: A practical introduction to statistics using R.* Cambridge: Cambridge University Press.
Baker, P., Hardie, A., & McEnery, T.（2006）. *A glossary of corpus linguistics.* Edinburgh: Edinburgh University Press.
Biber, D., Conrad, S., & Reppen, R.（1998）. *Corpus linguistics: Investigating language structure and use.* Cambridge: Cambridge University Press.
Bird, S., Klein, E., & Loper, E.（2009）. *Natural language processing with Python.* Sebastopol: O'Reilly.
Gries, S. Th.（2009）. *Quantitative corpus linguistics with R: A practical introduction.* New York: Routledge.
Gries, S. Th.（2010）. Methodological skills in corpus linguistics: A polemic and some pointers towards quantitative methods. In Harris, T., & Jaén, M. M.（Eds.）, *Corpus linguistics in language teaching*（pp. 121–146）. Frankfurt: Peter Lang.
Gries, S. Th.（2013）. *Statistics for linguistics with R: A practical introduction.* Second edition. Berlin: Mouton De Gruyter.
Hammond, M.（2002）. *Programming for linguists: JAVA™ technology for language research.* Oxford: Blackwell.
Hammond, M.（2003）. *Programming for linguists: Perl for language research.* Oxford: Blackwell.
石田基広・小林雄一郎（2013）.『Rで学ぶ日本語テキストマイニング』東京：ひつじ書房.
Jockers, M. L.（2014）. *Text analysis with R for students of literature.* New York: Springer.
Johnson, M.（2013）. *Essential Python for corpus linguistics.* Oxford: Blackwell.
小林雄一郎（2015）.「語彙多様性とリーダビリティを用いたテキスト分析」『外国語教育メディア学会中部支部外国語教育基礎研究部会2014年度報告論集』49–59.
Lander, J. P.（2013）. *R for everyone: Advanced analytics and graphics.* Boston: Addison-Wesley.
間瀬茂（2014）.『Rプログラミングマニュアル』第2版. 東京：数理工学社.
Mason, O.（2001）. *Programming for corpus linguistics: How to do text analysis with Java.* Edinburgh: Edinburgh University Press.
Matloff, N.（2011）. *The art of R programming: A tour of statistical software design.* Sebastopol: O'Reilly.
Meunier, F.（1998）. Computer tools for the analysis of learner corpora. In Granger, S.（Ed.）, *Learner English on computer*（pp. 19–37）. London: Longman.
荻野綱男・田野村忠温（編）（2012）.『Rubyによるテキストデータ処理』東京：明治書院.
大名力（2012）.『言語研究のための正規表現によるコーパス検索』東京：ひつじ書房.
斉藤常治・高橋佑幸（2014）.『PHPによる機械学習入門』東京：リックテレコム.
Sarkar, D.（2008）. *Lattice: Multivariate data visualization with R.* New York: Springer.
Schmitt, L. M., Christianson, K., & Gupta, R.（2010）. Linguistic computing with UNIX tools. In Kao, A., & Poteet, S. R.（Eds.）, *Natural language processing and text mining*（pp. 221–258）. New York: Springer.
Wickham, H.（2009）. *ggplot2: Elegant graphics for data analysis.* New York: Springer.

【Abstract】

Analyzing English Texts with R

Yuichiro KOBAYASHI

The purpose of the present paper is to introduce programming techniques for text mining. This paper explains how to make a concordance, wordlist, collocation table, and n-gram list, using R, a free software environment for data analysis. It also shows the visualization methods for text analysis, such as concordance plot and word cloud. By writing our own programs, we can conduct analyses that are not possible with ready-made tools, and tailor the output of the analyses to meet our research needs.